Homework 3

Due by **11:59pm, Thursday, September 27, 2018**.

Make sure you follow all the homework policies (../../policies/hw-policy.html).

All submissions should be done via Autolab (../../autolab.html).

The support page for matrix vector multiplication (../../../support/matrix-vect/index.html) should be very useful for this homework.

1. This might seem counter-intuitive but actually in this case looking at specific examples of these structured matrices might distract you from the high level structure in these matrices, which is more apparent in the formal definition. The final algorithm is easiest to design if you just concentrate on this abstract high level structure.

Sample Problem

The Problem

For this and the remaining problems, we will be working with $n \times n$ matrices (or two-dimensional arrays). So for example the following is a 3×3 matrix

$$\mathbf{M} = \begin{pmatrix} 1 & 2 & -3 \\ 2 & 9 & 0 \\ 6 & -1 & -2 \end{pmatrix}.$$

Given an $n \times n$ matrix $\mathbf{A}$, we will refer to the the entry in the ith row and jth column (for $0 \le i, j < n$) as $\mathbf{A}[i][j]$. So for example, for the matrix $\mathbf{M}$ above, $\mathbf{M}[1][2] = 0$. (We will use the convention that the top left element is the $[0][0]$ element.) We will also work with vectors of length n, which are just arrays of size n. For example the following is a vector of length 3

$$\mathbf{z} = \begin{pmatrix} 2 \\ 3 \\ -1 \end{pmatrix}.$$

As usual we will use $x[i]$ to denote the ith entry in the array/vector. For example, in the vector/array $\mathbf{z}$ above, we have $\mathbf{z}[2] = -1$.

We are finally ready to define the matrix-vector multiplication problem. Given an $n \times n$ matrix $\mathbf{A}$ and a vector $\mathbf{x}$ of length n, their product is denoted by

$$\mathbf{y} = \mathbf{A} \cdot \mathbf{x},$$

where $\mathbf{y}$ is also a vector of length n and its ith entry for $0 \le i < n$ is defined as follows:

$$\mathbf{y}[i] = \sum_{j=0}^{n-1} \mathbf{A}[i][j] \cdot \mathbf{x}[j].$$

For example, here is the worked out example for $\mathbf{M}$ and $\mathbf{z}$ above:

$$\begin{pmatrix} 1 & 2 & -3 \\ 2 & 9 & 0 \\ 6 & -1 & -2 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 3 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \times 2 + 2 \times 3 + (-3) \times (-1) \\ 2 \times 2 + 9 \times 3 + 0 \times (-1) \\ 6 \times 2 + (-1) \times 3 + (-2) \times (-1) \end{pmatrix} = \begin{pmatrix} 11 \\ 31 \\ 11 \end{pmatrix}.$$

Finally, we come to the actual question:

1. Design an $O(n^2)$ algorithm, which given any $n \times n$ matrix $\mathbf{A}$ and vector $\mathbf{x}$ of length n correctly computes $\mathbf{A} \cdot \mathbf{x}$.

   ### Note

   This is exactly Exercise 1 in the support page on matrix vector multiplication (../../../support/matrix-vect/index.html).

   ### Hint

   The obvious algorithm works for this problem.

2. Argue that your algorithm in part above runs in time $\Omega(n^2)$. (Hence, conclude that your algorithm runs in time $\Theta(n^2)$.)
3. (**Only think about this if you have time to spare.**) Argue that *any* algorithm that solves the matrix vector multiplication problem (for arbitrary $\mathbf{A}$ and $\mathbf{x}$) needs to take $\Omega(n^2)$ time.

   ### Note

   This is the same as Exercise 2 in the support page on matrix vector multiplication (../../../support/matrix-vect/index.html). Further, note that this part implies the second part but *if* you want to use third part to solve the second part, then you will have to present the solution to third part too. I would recommend that you solve part three only if you want to stretch your algorithmic thinking.

Click here for the Solution

Submission

You will **NOT** submit this question. This is for you to get into thinking more about matrix vector multiplication.

Question 1 (Programming Assignment) [30 points]

</> Note

This assignment can be solved in either Java, Python or C++ (you should pick the language you are most comfortable with). Please make sure to look at the supporting documentation and files for the language of your choosing.

The Problem

In this problem we will consider matrix and vector multiplication, but with the caveat that we already know something about the matrix. In particular for any $0 \le i, j \le n-1$, we have

$$\mathbf{U}_n[i][j] = \begin{cases} 0 & \text{if } j < i \\ 1 & \text{otherwise.} \end{cases}$$

If you "draw" this matrix you will note that all of the elements below the diagonal is zero: such matrices are called *upper triangular* matrices. Further, all non-zero values (in the "upper triangular" part) are all 1. Note that we only need to know the value of n to completely determine $\mathbf{U}_n$.

You have to write code to solve the following computational problem: given any integer $n \ge 1$ and a vector $\mathbf{x}$ of length n, return the vector $\mathbf{y} = \mathbf{U}_n \cdot \mathbf{x}$. Read on for more details on the format etc.

Sometimes it is easier to visualize a specific $\mathbf{U}_n$. Below is $\mathbf{U}_5$ written down in a 2D-array form:

```
1   1   1   1   1
0   1   1   1   1
0   0   1   1   1
0   0   0   1   1
0   0   0   0   1
```

Input

You will **not** receive the matrix as the input.

</> Note

You *could* build the matrix $\mathbf{U}_n$ yourself but this is **HIGHLY DISCOURAGED**. For the biggest input, you will need about 1 TB to store this matrix explicitly!

What you will receive is the vector itself. Each input file will contain one vector. The length of the vector will be on the first line of the file. The vector itself will take up the remaining lines. Once again, you will not have to handle the file reading and the creation of data structures. The packaged driver will handle that for you.

The input file can be summed up in the following format:

```
n
x0
x1
x2
.
.
.
xn-1
```

where each x_k is a value of the vector, in vertical order.

This is an example of what a file may look like, for size five:

```
  5
 43
234
-12
 32
-44
```

Output

The output will be the vector $\mathbf{y}$ in vertical order. I.e.

```
y0
y1
y2
.
.
.
yn-1
```

The correct result for the earlier example of $\mathbf{x}$ would be:

```
253
210
-24
-12
-44
```

Hint

The best possible algorithm for this problem runs in time $O(n)$.

! Note

Both the input and output parsers in each of the three languages are already written for you. Note that you have to work with the input data structures provided (which will come pre-loaded with the data from an input file).

! Addition is the only change you should make

Irrespective of what language you use, you will have to submit just one file. That file will come pre-populated with some stuff in it. You should **not** change any of those things because if you do you might break what the grader expects and end up with a zero on the question. You should of course add stuff to it (including helper functions and data structures as you see fit).

Java Python C++

Download Java Skeleton Code (HW3Java.zip)

Directory Structure

```
├── src
│   ├── ub
│       ├── cse
│           ├── algo
│               ├── Driver.java
│               ├── HW3Utility.java
│               ├── Solution.java
├── testcases/
│   ├── input1.txt
│   ├── input2.txt
│   ├── input5.txt
│   ├── output1.txt
│   ├── output2.txt
│   └── output5.txt
```

You are given two coding files: `Driver.java` and `Solution.java`. `Driver.java` takes the input file, parses it and creates an instance of the class `Solution` and prints the result to the command line. You only need to update the `Solution.java` file. You may write your own helper methods and data structures in it.

The testcases folder has 3 input files and their corresponding output files for your reference. We will use these three input files (and seven others) in our autograding.

Method you need to write:

```
5.    /**
       * This method must be filled in by you. You may add
       * but they must remain within the Solution class.
       * @return Your resulting vector.
       */
      public ArrayList<Integer> outputVector() {
          return new ArrayList<Integer>(); //Change this to
      }
```

The `Solution` class has 1 instance variable.

- `in_vector` which is an array of ints, and stores the input vector given by the driver.

Compiling and executing from command line:

Assuming you're in the same directory level as `src/` . Run `javac src/ub/cse/algo/*.java` to compile.

To execute your code on `input1.txt` , run `java -cp "src" ub.cse.algo.Driver testcases/input1.txt` . You can compare your output to `output1.txt` .

Submission

You only need to submit `Solution.java` to Autolab.

Grading Guidelines

We will follow the usual grading guidelines for programming questions (../../policies/hw-policy.html#grading).

Question 2 (Structured Upper Triangular matrix) [45 points]

The Problem

As we saw in the support page on matrix vector multiplication (../../../support/matrix-vect/index.html) (and in Question 1), if our matrix $\mathbf{A}$ is guaranteed to have some structure, then we can *potentially* solve the matrix-vector multiplication faster than the more general case of arbitrary $\mathbf{A}$ above. (Note that the vector $\mathbf{x}$ remains arbitrary though.) We explore this phenomenon more in this problem and the next.

Given a vector $\mathbf{r}$ of length n we define an $n \times n$ matrix $U^{\mathbf{r}}$ as follows. For any $0 \le i, j < n$:

$$U^{\mathbf{r}}[i][j] = \begin{cases} 0 & \text{if } j < i \\ r[i] & \text{otherwise.} \end{cases}$$

For example, if $\mathbf{r} = (6, 17, 9)$, we have

$$U^{(6,17,9)} = \begin{pmatrix} 6 & 6 & 6 \\ 0 & 17 & 17 \\ 0 & 0 & 9 \end{pmatrix}.$$

Given a vector $\mathbf{c}$ of length n we define an $n \times n$ matrix $U_{\mathbf{c}}$ as follows. For any $0 \le i, j < n$:

$$U_{\mathbf{c}}[i][j] = \begin{cases} 0 & \text{if } j < i \\ c[j] & \text{otherwise.} \end{cases}$$

For example, if $\mathbf{c} = (6, 17, 9)$, we have

$$U_{(6,17,9)} = \begin{pmatrix} 6 & 17 & 9 \\ 0 & 17 & 9 \\ 0 & 0 & 9 \end{pmatrix}.$$

Finally, we come to the two problems:

- Part `(a)` : Design an $O(n)$ algorithm, which given any two vectors $\mathbf{r}$ and $\mathbf{x}$ of length n correctly computes $U^{\mathbf{r}} \cdot \mathbf{x}$.

  Hint

  Try and see how you can compute $y[i+1]$ given $y[i]$ (where $\mathbf{y} = U^{\mathbf{r}} \cdot \mathbf{x}$.) Alternatively, if it helps, you can assume that there exists an algorithm to solve Question 1 and you can use it as a black box.

- Part `(b)` : This one has two sub-parts:
  1. Design an $O(n)$ algorithm, which given any two vectors $\mathbf{c}$ and $\mathbf{x}$ of length n correctly computes $U_{\mathbf{c}} \cdot \mathbf{x}$.

     Hint

     If it helps, you can assume that there exists an algorithm to solve Question 1 and you can use it as a black box.

  2. Argue that your algorithm in part `(b)` part 1 above runs in time $\Omega(n)$. (Hence, conclude that your algorithm runs in time $\Theta(n)$.)

Common Mistakes

- One common mistake is to present an algorithm for a specific family of vector $\mathbf{r}$ (or $\mathbf{c}$), e.g. $\mathbf{r}[i] = i$-- your algorithm should work on all possible vectors $\mathbf{r}$ and $\mathbf{x}$ for part `(a)` and for arbitrary vectors $\mathbf{c}$ and $\mathbf{x}$ for part `(b)` .
- If you are using an algorithm from Question 1 as a blackbox then in proving correctness of your algorithm for this question, you can only use the fact that the algorithm from Question 1 is correct (i.e. you cannot assume anything more specific about the algorithm for Question 1 other than the fact that it is correct). Also for Question 1 you can can only assume that you are given an algorithm with a runtime of $O(n)$-- but there is no $\Omega(\cdot)$ bound that you can assume for such an algorithm.

Walkthrough video

Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible.

! PDF only please

Autolab might not be able to display files in formats other than PDF (e.g. Word cannot be displayed). **If Autolab cannot display your file, then you will get a zero (0) on the entire question. Note that Autolab will NOT give an error message if you submit non-PDF file, so it is YOUR responsibility to make sure you submit in the correct format.** Also the file size has to be **at most 3MB**.

Grading Guidelines

We will follow the usual grading guidelines for non-programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to part `(a)` of this problem:

1. `Algorithm details` : `7` points.
2. `Big-Oh analysis` `3` points.

and here is the high level grading rubric for part `(b)` :

1. `Algorithm idea` : `15` points.
2. `Algorithm details` : `15` points.
3. `Big-Oh analysis` `2` points.

4. `Big-Omega analysis` `3` points.

`Note:` No proof idea or proof details of correctness is needed for this question. However your algorithm idea should implicitly include the proof idea of correctness. For an example of such a algorithm idea, see the algorithm idea for the structured matrix vector multiplication (for outer product) in the support page on matrix vector multiplication (../../../support/matrix-vect/index.html).

! Note

You will get a zero on the entire problem if you present an $\Omega(n^2)$ time algorithm.

! Note

If you do not have separated out algorithm details and runtime analysis for part `(a)` and algorithm idea, algorithm details, Big-Oh analysis and Big-Omega analysis for part `(b)`, you will get a zero(0) in the corresponding parts irrespective of the technical correctness of your solution.

Templates

Download LaTeX template. (latex-template.zip)

Download Microsoft Word template. (word-template.zip)

! Note

Your must explicitly list your sources and collaborators when you upload your submission to Autolab. Note that there are only five allowed sources (../../policies/hw-policy.html). If you have used a source that is not allowed, please do **not** submit your homework. If you did not consult any source or did not collaborate with anyone just say `None` .

Questions 3 (Approaching FFT) [25 points]

The Problem

For this problem we will assume that $n = m^2$ for some integer $m \geq 2$. Further for any integers $0 \leq i, j < n$ we will denote their m-ary representations by (i_0, i_1) and (j_0, j_1) (where $i_0, i_1, j_0, j_1 \in \{0, \ldots, m-1\}$). In other words,

$$i = i_0 + i_1 \cdot m,$$

and

$$j = j_0 + j_1 \cdot m.$$

We are now ready to define our structured matrix $\mathbf{A}$. For every k, ℓ such that $0 \le k + \ell \le 1$, we are given $m \times m$ matrices $\mathbf{B}^{k,\ell}$. Then the matrix $\mathbf{A}$ is defined as follows. For every $0 \le i, j < n$,

$$\mathbf{A}[i][j] = \prod_{k,\ell : 0 \le k+\ell \le 1} \mathbf{B}^{k,\ell}[i_k][j_\ell].$$

Let us first illustrate the definition above with an example. Let $m = 2$. In this case say we are given the following three matrices

$$\mathbf{B}^{0,0} = \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix},$$

$$\mathbf{B}^{0,1} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix},$$

and

$$\mathbf{B}^{1,0} = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}.$$

Then the final matrix is

$$\mathbf{A} = \begin{pmatrix} 1\cdot 1\cdot 5 & 1\cdot 1\cdot 6 & 1\cdot 2\cdot 5 & 1\cdot 2\cdot 6 \\ 2\cdot 3\cdot 5 & 2\cdot 3\cdot 6 & 2\cdot 4\cdot 5 & 2\cdot 4\cdot 6 \\ 1\cdot 1\cdot 7 & 1\cdot 1\cdot 8 & 1\cdot 2\cdot 7 & 1\cdot 2\cdot 8 \\ 2\cdot 3\cdot 7 & 2\cdot 3\cdot 8 & 2\cdot 4\cdot 7 & 2\cdot 4\cdot 8 \end{pmatrix} = \begin{pmatrix} 5 & 6 & 10 & 12 \\ 30 & 36 & 40 & 48 \\ 7 & 8 & 14 & 16 \\ 42 & 48 & 56 & 64 \end{pmatrix},$$

where in the above, we index the rows of $\mathbf{A}$ by $(i_0, i_1) \in \{0,1\}^2$ and are ordered from "top" to "bottom" as $(0,0), (0,1), (1,0), (1,1)$ while the columns of $\mathbf{A}$ are indexed by $(j_0, j_1) \in \{0,1\}^2$ and are ordered from "left" to "right" as $(0,0), (0,1), (1,0), (1,1)$.

Let's unpack this a bit

Just to make sure everyone is on the same page, let me walk through one of the calculations above. Say we are interested in $\mathbf{A}[2][1]$, i.e. $i = 2$ and $j = 1$. Then we consider the 2-ary (or binary) representation of $i = (i_1, i_0) = (1, 0)$ and $j = (j_1, j_0) = (0, 1)$. In other words, we have $i_0 = 0, i_1 = 1, j_0 = 1$ and $j_1 = 0$. Now note that by definition we have

$$\mathbf{A}[i][j] = \mathbf{B}^{0,0}[i_0][j_0] \cdot \mathbf{B}^{0,1}[i_0][j_1] \cdot \mathbf{B}^{1,0}[i_1][j_0].$$

In particular, we have

$$\mathbf{A}[2][1] = \mathbf{B}^{0,0}[0][1] \cdot \mathbf{B}^{0,1}[0][0] \cdot \mathbf{B}^{1,0}[1][1].$$

Now, note that we have $\mathbf{B}^{0,0}[0][1] = 1$, $\mathbf{B}^{0,1}[0][0] = 1$ and $\mathbf{B}^{1,0}[1][1] = 8$ and thus, we have

$$\mathbf{A}[2][1] = 1 \cdot 1 \cdot 8 = 8,$$

as stated above.

It is legitimate to wonder why should one look at this somewhat complicated way of defining such a matrix. Turns out that this family of matrices contains perhaps the most widely used structured matrix: the so called discrete Fourier matrix. (See the section on *Discrete Fourier Transform* in support page on matrix vector multiplication (../../../support/matrix-vect/index.html) for more on these matrices including their definition.) Interestingly, this is probably one of the very few cases in CSE 331 that looking at a definition abstractly *might* be more beneficial then trying out some examples. The latter is definitely true when trying to design the algorithm in the part `(b)` below. [1]

- Part `(a)` : As a warmup, you will argue a re-formulation of the matrix-vector multiplication in a slightly different syntactic format-- this would be useful in proving part `(b)` .

  Think of both $\mathbf{x}$ and $\mathbf{y}$ as an $m \times m$ matrix instead of just a traditional one-dimensional array of length $n = m^2$. So e.g. let $\mathbf{X}$ be the $m \times m$ matrix such that $\mathbf{x}[i] = \mathbf{X}[i_0][i_1]$. Similarly, let $\mathbf{Y}$ be the $m \times m$ matrix such that $\mathbf{y}[i] = \mathbf{Y}[i_0][i_1]$. Finally, define a 4-dimensional $m \times m \times m \times m$ structure $\mathbf{A}'$, which is defined as follows:

  $$\mathbf{A}'[i_0][i_1][j_0][j_1] = \mathbf{A}[i][j].$$

  Now, you want to argue that for any $i_0, i_1 \in \{0, \ldots, m-1\}$, we have

  $$\mathbf{Y}[i_0][i_1] = \sum_{j_0=0}^{m-1}\sum_{j_1=0}^{m-1} \mathbf{A}'[i_0][i_1][j_0][j_1] \cdot \mathbf{X}[j_0][j_1].$$

- Part `(b)` : So now to the actual problem. Design an algorithm that given arbitrary matrices $\mathbf{B}^{k,\ell}$ for every k, ℓ such that $0 \le k + \ell \le 1$ and an arbitrary vector $\mathbf{x}$ of length n, computes $\mathbf{A} \cdot \mathbf{x}$, where $\mathbf{A}$ is as defined above. Your algorithm should run in $O(nm) = O(n^{3/2})$ time.

  Hint

  The *distributive law strategy* outlined in support page on matrix vector multiplication (../../../support/matrix-vect/index.html) does lead to the desired algorithm. However, note that the above question is bit more involved than the example use of the distributive law strategy outlined in support page on matrix vector multiplication (../../../support/matrix-vect/index.html).

Note

it turns out that the above idea can be generalized to the case when $n = 2^m$ and results in an $O(n \log n)$ time algorithm to compute the DFT. However, you only need to show the weaker statement above.

Walkthrough video

Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible.

! PDF only please

Autolab might not be able to display files in formats other than PDF (e.g. Word cannot be displayed). **If Autolab cannot display your file, then you will get a zero (0) on the entire question. Note that Autolab will NOT give an error message if you submit non-PDF file, so it is YOUR responsibility to make sure you submit in the correct format.** Also the file size has to be **at most 3MB**.

Grading Guidelines

We will follow the usual grading guidelines for non-programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to part `(a)` of this problem:

1. `Proof Idea` : `7` points.
2. `Proof Details` `3` points.

and here is the high level grading rubric for part `(b)` :

1. `Algorithm idea` : `6` points.
2. `Algorithm details` : `6` points.
3. `Proof idea` : `1` points for a proof idea that your algorithm is correct (proof details are not required).
4. `Runtime analysis` : `2` points for showing that your algorithm runs in $O(m^3)$ time.
5. `Note:` You will get a zero if your algorithm's runtime is $\Omega(n^2) = \Omega(m^4)$.

! Note

If you do not have separated out proof idea/details for part `(a)` and proof/algorithm idea, algorithm details and runtime analysis for part `(b)`, you will get a zero(0) in the corresponding parts irrespective of the technical correctness of your solution..

Templates

Download LaTeX template. (latex-template.zip)

Download Microsoft Word template. (word-template.zip)

! Note

Your must explicitly list your sources and collaborators when you upload your submission to Autolab. Note that there are only five allowed sources (../../policies/hw-policy.html). If you have used a source that is not allowed, please do **not** submit your homework. If you did not consult any source or did not collaborate with anyone just say `None`.